Burnett, Andrew W. and Parkes, Andrew J. (2016) Systematic search for local-search SAT heuristics. In: 6th International Conference on Metaheuristics and Nature Inspired Computing (META 2016), 27-31 Oct 2016, Marrakech, Morocco.

Systematic Search for Local-Search SAT Heuristics

Andrew W. Burnett and Andrew J. Parkes

ASAP Research Group
School of Computer Science, University of Nottingham, United Kingdom
psxab4 @nottingham.ac.uk, ajp @cs.nott.ac.uk

Abstract. Heuristics for local-search are a commonly used method of improving the performance of algorithms that solve hard computational problems. Generally these are written by human experts, however a long-standing research goal has been to automate the construction of these heuristics. In this paper, we investigate the applicability of a systematic search on the space of heuristics to be used in a local-search SAT solver.

1 Introduction

Algorithms to solve hard computational problems generally employ carefully-designed heuristics to improve their performance. For example, within local-search procedures for propositional satisfiability (SAT)[1]. A well-established class of algorithms are those inspired by "WalkSAT" [5, 4, 2], which begins with an assignment of variables, and in each iteration, a selected variable is chosen to have its assignment changed. It is the role of the "Variable Selection Heuristic" (VSH) to choose this variable on each iteration. Usually, the VSH is designed by a human expert; however, a long standing desire has been to automate the generation of heuristics to reduce the human element, and one well-established technique for this is using genetic programming (GP). Specifically, in the context of SAT, Fukunaga [1] used GP to build VSHs for use in a local-search SAT solver; the GP was able to build VSHs that performed competitively with two of the best heuristics at the time.

Separately, Katayama [3] observed that, within the context of functional programming, when given an initial function set, desired type signature, input and output, a brute-force search of the space of procedures was a viable method for finding programs that satisfy these constraints. In this paper, we observe that the space of potential VSHs for SAT can be small enough that it can be amenable to systematic search methods. We used a tree-based systematic search (to a certain depth) to find candidate VSHs. We show that this method is able to find well-performing VSHs, and so argue for a role for systematic search to complement sampling based methods such as GP.

2 Language used, experiments and results

Fukunaga [1] outlined several quantities associated with variables and clauses present in current heuristics as follows:

- *Broken Clause* - A clause that is broken (evaluates to $False$ under the current assignment)
- *Positive Gain* - Number of unsatisfied clauses that will become satisfied if a variable v is flipped
- *Negative Gain* - Number of satisfied clauses that will become unsatisfied if a variable v is flipped
- *Net Gain* - The net change in number of unsatisfied clauses if a variable v is flipped
- *Age* - The number of iterations since a variable v was last flipped

Figure 1 outlines the grammar used to generate VSHs. Space precludes giving full details, but we work with a similar language to Fukunaga [1]. Examples of VSHs defined using this grammar are given in Figure 3. Using this grammar we enumerated all possible VSHs that contain 15 or fewer symbols (we regard the number of symbols as the "depth" of the VSH). We chose 15 as it is just beyond depth 14, where an example of an already known well-performing WalkSAT VSH is

[1] SAT is the decision problem of whether a given propositional logic formula P, has a truth assignment to the variables in P such that P is satisfied. Generally, P is given in Clausal Normal Form (CNF), that is a conjunction of clauses, each clause is a disjunction of literals, and a literal is variable or its negation.

⟨*VSH*⟩ ::= IfRandLt ⟨*Probability*⟩ ⟨*VSH*⟩ ⟨*VSH*⟩
| GetBestVar ⟨*GainType*⟩ ⟨*VarSet*⟩
| GetSecondBestVar ⟨*GainType*⟩ ⟨*VarSet*⟩
| GetOldestVar ⟨*VSH*⟩ ⟨*VSH*⟩
| IfTabu ⟨*Age*⟩ ⟨*VSH*⟩ ⟨*VSH*⟩
| IfVarCompare ⟨*Comparator*⟩ ⟨*GainType*⟩ ⟨*VSH*⟩ ⟨*VSH*⟩
| IfVarCond ⟨*Comparator*⟩ ⟨*GainType*⟩ ⟨*Integer*⟩ ⟨*VSH*⟩ ⟨*VSH*⟩
| VarRandom ⟨ *VarSet* ⟩
| IfNotMinAge ⟨*VarSet*⟩ ⟨*VSH*⟩ ⟨*VSH*⟩
⟨*VarSet*⟩ ::= BrokenClause0
⟨*Comparator*⟩ ::= LessThan | LessThanEqual | Equal
⟨*GainType*⟩ ::= PosGain | NegGain | NetGain
⟨*Integer*⟩ ::= -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5
⟨*Age*⟩ ::= 5 | 10 | 20 | 30 | 40 | 50
⟨*Probability*⟩ ::= 0.1 | 0.3 | 0.5 | 0.7 | 0.9

Fig. 1. The grammar used to generate candidate VSHs. Compared to Fukunaga's work [1] we reduce the search space by only considering a single broken clause, *BrokenClause0*.

Table 1. A numerical analysis of the VSH at various depths, D. The ratio of numbers at depth n versus $n-1$; number with fitness ≥ 20, and the % with fitness ≥ 20

D	# VSP	ratio	≥ 20	% ≥ 20
5	1	-	0	0
6	24	24	0	0
7	189	7.9	7	3.7
8	614	3.2	30	4.9
9	1,272	2.1	0	0
10	3,996	3.1	6	0.2
11	12,173	3.0	380	3.1
12	62,238	5.1	3,742	6.0
13	223,155	3.6	11,708	5.2
14	714,542	3.2	23,384	3.3
15	2,264,475	3.2	58,161	2.6

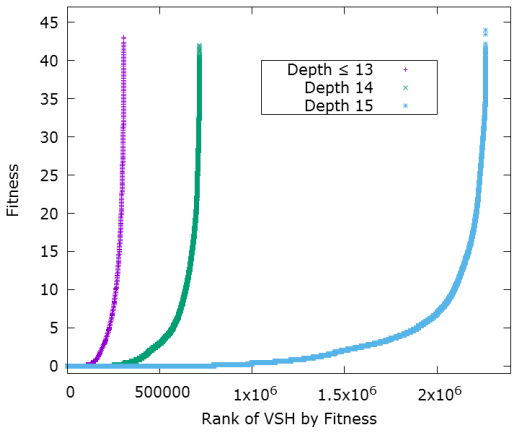

Fig. 2. Shows all heuristics at each level and their fitness sorted in ascending order

found. In Table 1 we can see that the number of VSHs grows rapidly with the number of symbols; tending towards increasing by around a factor of 3.2 per extra symbol. Note that the number of VSHs is still very small when compared to the size of the search space of most combinatorial optimisation problems. Hence, if the fitness of a VSH could be determined quickly, then the search would be easy; however, evaluating the fitness of a VSH requires testing it against several problem instances. In our case, this is 100 satisfiable SAT instances from SATLIB [2]. In an attempt to reduce computation time, not all VSHs were ran on all instances; rather, if a VSH performed poorly on an initial selection, it was not ran on the remaining instances. We repeated this experiment five times for each heuristic and, taking the number of times a solution was successfully found and the number of flips required to obtain these solutions, we computed a fitness value for a VSH. (details not given due to lack of space.) It should be noted that this fitness value should be considered "noisy" due, in part, to the random assignment initially generated for each VSH on each training instance.

In Table 1 we can see the number and % of VSHs at each depth with a score greater than 20; a value chosen because it is close to the fitness value of the WalkSAT VSH. We find these values to be surprisingly large. Figure 2 shows each heuristic, ordered by fitness, plotted against its fitness. We can see from this graph that a large proportion of these VSHs generated are unable to solve a single problem instance in our training set. Figure 3 shows two examples of VSHs written using the grammar in Figure 1; WalkSAT and VSH-1. VSH-1 is the VSH that scored highest according to our fitness function. This Figure also shows the results of running these two VSHs on satisfiable

[2] `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

```
WalkSAT(0.5):
 IfVarCond NegGain == 0
   (GetBestVar NegGain BC0)
   (IfRandLt 0.5
     (GetBestVar NegGain BC0)
     (VarRandom BC0))
```

```
VSH-1:
 GetOldestVar
   (GetOldestVar
     (GetOldestVar
       (GetBestVar NetGain BC0)
       (GetBestVar NegGain BC0))
     (GetBestVar NetGain BC0))
   (GetBestVar PosGain BC0)
```

	uf50		uf100		uf150		uf200		uf250	
Name	SR	AFS	SR	AFS	SR	AFS	SR	AFS	SR	AFS
WalkSAT	**1.0**	**790**	**0.998**	**4036**	0.95	11833	0.853	11550	0.858	22514
VSH-1	0.998	319	0.994	1649	**0.975**	**4357**	**0.933**	**7500**	**0.957**	**13706**

Fig. 3. Top Left: WalkSAT (at noise 0.5). Due to redundancies in the language, for each WalkSAT VSH with noise p, there are two semantically identical VSH produced by changing *Equal* to *LessThanEqual*. Top Right: VSH-1. Bottom: Results of running the heuristics VSH-1 and WalkSAT on the testing sets. Showing the median Success Rate (SR) of instances solved and Average Flips to Solution(AFS)

instances from SATLIB not included in our training set. We can see that, although WalkSAT has a higher success rate on easier problem sets, conversely, on harder problem sets VSP-1 outperforms WalkSAT in both success rate and number of flips. Suggesting that, while our fitness function is noisy, it is a good indicator of whether a VSH performs well for problems of larger size.

3 Conclusions and Future Work

We have given a method to automatically build effective heuristics, with the aim to save time of human experts and potentially allowing partial specialisation to particular sets of problem instances. The crucial observation is that, with reasonable restriction of the language, the space of "reasonable" VSHs is surprisingly small and, as a proportion, the number of "good" VSHs is surprisingly large. This allowed us to search the space using a systematic search and produce a number of VSHs that outperform the original WalkSAT on some training sets (we do not claim here that VSH-1 is more effective across a wider set of instances, only that it worked well on the instances used.) Besides avoiding repetition, a systematic search has the potential to discover 'isolated heuristics' that might not be easy to find using GP; some good heuristics could potentially have substructures that are not useful in general and so could be less likely to emerge in GP.

Future work should reduce the search space by taking account of semantic equivalence - VSHs that are syntactically different but semantically equivalent need to be recognised as such. Another potential future direction is in identifying poorly performing VSHs and not having to evaluate these at all using a surrogate fitness functions. In such systematic search a natural issue is what search ordering to use; in this preliminary study we have simply evaluated all within the depth limit; but a heuristic ordered "best first" should be considered using current knowledge about good VSHs leading to a form of large-scale neighbourhood search, possibly hybridising or complementing GP methods.

Acknowledgements: Andrew W. Burnett thanks the EPSRC for financial support.

References

1. Alex S Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.
2. Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2004.
3. Susumu Katayama. Power of brute-force search in strongly-typed inductive functional programming automation. In *PRICAI 2004: Trends in Artificial Intelligence*, pages 75–84. Springer, 2004.
4. David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. 1997.
5. Bart Selman, Henry A Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 337–343. American Association for Artificial Intelligence, 1994.